REMARKS

The above amendment and these remarks are responsive to the communication from Examiner Majid Banankhah mailed 19 Oct 2000, rejecting claims 1-8, all of the claims in the case.

DRAWINGS

The communication from the Examiner refers to a form PTO-948 concerning notice of draftsman's patent drawing review. However, no such form was included in the communication, nor identified in the Office Action Summary. However, applicant will submit formal drawings in due course.

TITLE

The Examiner requires a new title. Applicant has amended the specification and abstract accordingly.

Claims 1-8 have been rejected under 35 U.S.C. 103 over Davidson et al. (U. S. Patent 5,630,136), in view of Periwal et al. (U. S. Patent 5,644,768).

Applicants traverse with respect to claims 2-3 and 8, and has amended the other claims to clarify the distinctions with respect to the cited art.

Davidson and Periwall relate, respectively, to the OS primitive for locking such as mutex or sleep/wake. Typically, OS mutex primitives simultaneously wake all threads which are sleeping/waiting on a mutex-protected resources. It is then happenstance which thread actually makes it first to access the protected resource (such as by successfully obtaining Davidson's "baton"), and the others must go back to sleep/wait state. If, for example, 15 threads are waiting on a resource. As one becomes free, all 15 threads are awakened, one grabs the baton, and the other 14 all must return to wait state. This behavior causes thrashing on the multi-tasking OS's system "run" queue, where threads that are runnable, or waiting to be runnable, are tracked. By the time the last of the 15 threads is

satisfied, it has make a lot of unsuccessful attempts to be serviced. A great deal of system resources is expended in managing such a run/wait queue.

On the other hand, applicant has provided a stationary queue. Such a queue, which includes two counters, insures not only fairness, but that only one thread at a time is awakened and given access, so system performance in resource constrained scenarios is maintained. There is no thrashing of a run/wait queue in the kernel.

Periwall, on the other hand, teaches a mutex (record) and a nesting mechanism for preventing deadlocks when accessing the mutex. This is done to prevent deadlock conditions but, as with Davidson, multiple threads are awakened simultaneously.

As the Examiner states, Davidson does fail to explicitly teach of a queue for allocating access to resources. That is the essence of applicant's invention. And yet, applicant does such without a queue, but rather with a stationary queue (which he explains at Page 7 lines 14-15 includes a counter pair). Thus, applicant's invention achieves queue-like behavior (first-come, first-served) but without the use of a real queue.

EN995139   5   S/N 08/820,181

Claim 1 has been amended to clarify that the requesting threads are animated one-by-one in order of request.

Claim 2, at lines 9 and 10, specifies that "a next thread in line is to be re-animated and grated access", and thus distinguishes Davidson and Periwall.

Claim 3 depends from claim 2.

Claim 4 has been amended to clarify that a thread is awakened and granted access to a resource in fair order.

Claims 5-7 have been amended to clarify that requesting threads are animated one-by-one in order of request, or in fair order.

Claim 8, at lines 10 and 11, already carries the essence of the above noted distinctions with respect to Davidson and Periwall.

SUMMARY AND CONCLUSION

Applicant urges that the above amendments be entered and the case passed to issue with claims 1-8.

If, in the opinion of the Examiner, a telephone conversation with applicant's attorney could possibly facilitate prosecution of the case, he may be reached at the number noted below.

Sincerely,

G. W.Wilhelm, Jr.

By

Shelley M Beckstrand
Reg. No. 24,886

Date: 19 Jan 2001

Shelley M Beckstrand, P.C.
Attorney at Law
314 Main Street
Owego, NY 13827

Phone: (607) 687-9913
Fax: (607) 687-7848

a:amend.wpd

CLEAN SET OF CLAIMS

We claim:

1. A multi-tasking operating system for managing
simultaneous access to scarce or serially re-usable
resources by multiple process threads, comprising:

    at least one resource;

    a plurality of threads requesting access to said
    resource; and

    a stationary queue for allocating access to said
    resource amonst said threads one-by-one in order of
    request.

2. A multi-tasking operating system stationary queue for
managing simultaneous access to scarce or serially re-usable
resources by multiple process threads, the stationary queue
comprising:

    a sleep code routine for generating a unique block

6   identifier when a process thread temporarily cannot

7   gain access to said resource and must be suspended; and

8   a wakeup code routine for generating a unique run

9   identifier when a next thread in line is to be

10   re-animated and granted access to said resource.

11   3.   The system of claim 2, further comprising:

12   a wait counter for counting the cumulative number of

13   threads that have been temporarily denied the resource;

14   a satisfied counter for counting the cumulative number

15   of threads that have been denied access and

16   subsequently granted access to said resource;

17   said sleep code routine being responsive to said wait

18   counter for generating said run identifier; and

19   said wakeup code routine being responsive to said

20   satisfied counter for generating said run identifier.

1   4.   A method for managing simultaneous access to scarce or

2   serially re-usable resources by multiple process threads,

3   comprising the steps of:

4 responsive to a request from a thread for a resource

5 which is not available, creating a block identifier

6 based on the number of threads temporarily denied the

7 resource; and

8 blocking said thread using said block identifier,

9 thereby enabling subsequent wake up of said thread in

10 fair order after said number of threads have been

11 serviced.

1 5. A method for managing simultaneous access to scarce or

2 serially re-usable resources by multiple process threads,

3 comprising the steps of:

4 responsive to a resource becoming available, creating a

5 run identifier based on the number of threads that have

6 been first forced to wait and have been subsequently

7 satisfied; and

8 awakening and running one next thread using said run

9 identifier, thereby granting access to said resource by

10 said process threads one-by-one in fair order.

1   6.   A method for managing simultaneous access to scarce or

2   serially re-usable resources by multiple process threads,

3   comprising the steps of:

4   responsive to a request for a resource which is not

5   available,

6   creating a block identifier based on the number of

7   threads temporarily denied the resource; and

8   blocking the thread using said block identifier;

9   and

10

11   responsive to a resource becoming available,

12   creating a run identifier based on the number of

13   threads that have been first forced to wait and

14   have been subsequently satisfied; and

15   awakening and running a next thread in fair order

16   using said run identifier.

1   7.   a memory device for storing signals for controlling the

2   operation of a computer to manage simultaneous access to

3   scarce or serially re-usable resources by multiple process

4   threads, according to the steps of

5   responsive to a request for a resource which is not

6   available,

7   creating a block identifier based on the number of

8   threads temporarily denied the resource; and

9   blocking the thread using said block identifier;

10   and

11

12   responsive to a resource becoming available,

13   creating a run identifier based on the number of

14   threads that have been first forced to wait and

15   have been subsequently satisfied; and

16   awakening and running the one next thread in fair

17   order using said run identifier.

1   8.   A memory device for storing signals to structure

2   the components of a digital computer to form a stationary

3   queue for managing simultaneous access to scarce or serially

4   re-usable resources by multiple process threads, comprising:

5   a sleep code routine for generating a unique block

6   identifier when a process thread temporarily cannot

7   gain access to said resource and must be suspended;

8

9   a wakeup code routine for generating a unique run

10   identifier when a next thread in line is to be

11   re-animated and granted access to said resource;

12   a wait counter for counting the cumulative number of

13   threads that have been temporarily denied the resource;

14   a satisfied counter for counting the cumulative number

15   of threads that have been denied access and

16   subsequently granted access to said resource;

17   said sleep code routine being responsive to said wait

18   counter for generating said run identifier; and

19   said wakeup code routine being responsive to said

20   satisfied counter for generating said run identifier.

MARKED UP CLAIMS

We claim:

1   1.   [Amended]   A multi-tasking operating system for

2   managing simultaneous access to scarce or serially re-usable

3   resources by multiple process threads, comprising:

4   at least one resource;

5   a plurality of threads <u>requesting access to said</u>

6   <u>resource</u>; and

7   a stationary queue for allocating access to said

8   resource amonst said threads <u>one-by-one in order of</u>

9   <u>request</u>.

1   2.   A multi-tasking operating system stationary queue for

2   managing simultaneous access to scarce or serially re-usable

3   resources by multiple process threads, the stationary queue

4   comprising:

5   a sleep code routine for generating a unique block

6   identifier when a process thread temporarily cannot

7   gain access to said resource and must be suspended; and

8   a wakeup code routine for generating a unique run

9   identifier when a next thread in line is to be

10   re-animated and granted access to said resource.

11   3.   The system of claim 2, further comprising:

12   a wait counter for counting the cumulative number of

13   threads that have been temporarily denied the resource;

14   a satisfied counter for counting the cumulative number

15   of threads that have been denied access and

16   subsequently granted access to said resource;

17   said sleep code routine being responsive to said wait

18   counter for generating said run identifier; and

19   said wakeup code routine being responsive to said

20   satisfied counter for generating said run identifier.

1   4.   [Amended]   A method for managing simultaneous access to

2   scarce or serially re-usable resources by multiple process

3   threads, comprising the steps of:

4   responsive to a request _from a thread_ for a resource

5   which is not available, creating a block identifier

6   based on the number of threads temporarily denied the

7   resource; and

8   blocking [the] said thread using said block identifier,

9   thereby enabling subsequent wake up of said thread in

10   fair order after said number of threads have been

11   serviced.

1   5.   [Amended] A method for managing simultaneous access to

2   scarce or serially re-usable resources by multiple process

3   threads, comprising the steps of:

4   responsive to a resource becoming available, creating a

5   run identifier based on the number of threads that have

6   been first forced to wait and have been subsequently

7   satisfied; and

8   awakening and running [the] one next thread using said

9   run identifier, thereby granting access to said

10   resource by said process threads one-by-one in fair

11   order.

1   6.   [Amended] A method for managing simultaneous access to

2   scarce or serially re-usable resources by multiple process

3   threads, comprising the steps of:

4   responsive to a request for a resource which is not

5   available,

6   creating a block identifier based on the number of

7   threads temporarily denied the resource; and

8   blocking the thread using said block identifier;

9   and

10

11   responsive to a resource becoming available,

12   creating a run identifier based on the number of

13   threads that have been first forced to wait and

14   have been subsequently satisfied; and

15   awakening and running [the] a next thread in fair

16   order using said run identifier.   .

1   7.   [Amended]   A memory device for storing signals for

2   controlling the operation of a computer to manage

3   simultaneous access to scarce or serially re-usable

4   resources by multiple process threads, according to the

5   steps of

6   responsive to a request for a resource which is not

7   available,

8   creating a block identifier based on the number of

9   threads temporarily denied the resource; and

10   blocking the thread using said block identifier;

11   and

12

13   responsive to a resource becoming available,

14   creating a run identifier based on the number of

15   threads that have been first forced to wait and

16   have been subsequently satisfied; and

17   awakening and running the one next thread in fair

18   order using said run identifier.

1   8.   A memory device for storing signals to structure

2   the components of a digital computer to form a stationary

3 queue for managing simultaneous access to scarce or serially

4 re-usable resources by multiple process threads, comprising:

5 a sleep code routine for generating a unique block

6 identifier when a process thread temporarily cannot

7 gain access to said resource and must be suspended;

8

9 a wakeup code routine for generating a unique run

10 identifier when a next thread in line is to be

11 re-animated and granted access to said resource;

12 a wait counter for counting the cumulative number of

13 threads that have been temporarily denied the resource;

14 a satisfied counter for counting the cumulative number

15 of threads that have been denied access and

16 subsequently granted access to said resource;

17 said sleep code routine being responsive to said wait

18 counter for generating said run identifier; and

19 said wakeup code routine being responsive to said

20 satisfied counter for generating said run identifier.

[STATIONARY QUEUE FOR SCARCE RESOURCE MANAGEMENT]

SYSTEM AND METHOD FOR QUEUE-LESS ENFORCEMENT OF QUEUE-LIKE

BEHAVIOR ON MULTIPLE THREADS ACCESSING A SCARCE RESOURCE

Background of the Invention

Technical Field of the Invention

This invention relates to a method and system for

managing shared resources, and more particularly to a

stationary queue for managing access to scarce or serially

reusable resources by multiple process threads.

SYSTEM AND METHOD FOR QUEUE-LESS ENFORCEMENT OF QUEUE-LIKE BEHAVIOR ON MULTIPLE THREADS ACCESSING A SCARCE RESOURCE

Background of the Invention

Technical Field of the Invention

This invention relates to a method and system for managing shared resources, and more particularly to a stationary queue for managing access to scarce or serially reusable resources by multiple process threads.

[STATIONARY QUEUE FOR SCARCE RESOURCE MANAGEMENT]

SYSTEM AND METHOD FOR QUEUE-LESS ENFORCEMENT OF QUEUE-LIKE

BEHAVIOR ON MULTIPLE THREADS ACCESSING A SCARCE RESOURCE

Abstract of the Disclosure

A system and method for managing simultaneous access to a scarce or serially re-usable resource by multiple process threads. A stationary queue is provided, including a wait counter for counting the cumulative number of threads that have been temporarily denied the resource; a satisfied counter for counting the cumulative number of threads that have been denied access and subsequently granted access to said resource; a sleep code routine responsive to the wait counter for generating a run identifier; and a wakeup code routine responsive to the satisfied counter for generating the run identifier.

SYSTEM AND METHOD FOR QUEUE-LESS ENFORCEMENT OF QUEUE-LIKE BEHAVIOR ON MULTIPLE THREADS ACCESSING A SCARCE RESOURCE

Abstract of the Disclosure

A system and method for managing simultaneous access to a scarce or serially re-usable resource by multiple process threads. A stationary queue is provided, including a wait counter for counting the cumulative number of threads that have been temporarily denied the resource; a satisfied counter for counting the cumulative number of threads that have been denied access and subsequently granted access to said resource; a sleep code routine responsive to the wait counter for generating a run identifier; and a wakeup code routine responsive to the satisfied counter for generating the run identifier.